計算機アーキテクチャ　96－16
（1992. 10. 23）

機器組込みマイコン用最適化Cコンパイラの性能評価

高山秀一　湯川博司　漆原誠一
田中旭　富永宣輝　入交旬子

松下電器産業（株）情報システム研究所

あらまし　機器組み込みマイコンのプログラムサイズは年々増加している。従来のようにアセンブラ言語で開発することが困難になり、高級言語での開発が望まれている。しかし高級言語で開発した場合、プログラムのサイズは大きく、実行速度も遅くなる。機器組み込み用途の場合、特にプログラムサイズは小さく、実行速度も高速でなければならないという要求に応えるため、コンパイラの生成コードを徹底的に分析してMN10200マイコンアーキテクチャを設計した。本稿では、コンパイラがアーキテクチャをどのように利用して生成コードサイズ縮小、及びプログラムの実行速度の高速化を達成するか生成コードの例を参照して考察し評価する。

和文キーワード　コンパイラ　最適化　アーキテクチャ　MN10200

AN EVALUATION OF THE OPTIMIZING C COMPILER WHICH APPLIES TO ERECTRICAL PRODUCTS

Shuichi Takayama　Hiroshi Yukawa　Seiichi Urushibara
Akira Tanaka　Nobuki Tominaga　Junko Irimajiri

Information Systems Research Laboratory, Matsushita Electric Industrial Co.,Ltd.

Abstract

Recently, as the size of program which applies to electrical profducts increses, such a program is expected to be developed by the high-level language instead of the assembly language. But as far, the performance of program developed by the high-level language is not sufficient. So, we developed the architecture of the micro computer named "MN10200" and its C compiler in consideration of the codes genarated by C compiler in order to solve this problem. We describe and evaluate how this compiler use its architecture taking examples.

英文 key words　compiler　optimizing　architecture　MN10200

1 はじめに

機器組み込みマイコンのプログラム開発においては、ROM 化の際のコストを低減するためプログラムサイズはできるだけ小さいほうがよい。

従来、機器組み込みマイコンのプログラムは比較的小規模であったし、コンパイラが生成するコードはアセンブラ言語で開発されたものに比べてサイズが大きく、実行速度も遅かったので、開発にはアセンブラ言語が使用されてきた。

しかし近年、機器組み込み分野においてもプログラムの規模は増大する傾向にあり、従来のようなアセンブラ言語での開発は困難になってきた。このような問題点に立脚し、

- プログラムサイズの縮小
- 実行速度の高速化
- プロセッサ資源の節約

を果たすため、コンパイラの生成コードを分析し、無駄なハードウエア資源を省いた高級言語指向マイコン MN10200[1] と最適化コンパイラを開発した。

本稿では、MN10200 用コンパイラがそのアーキテクチャの特徴を活用し、プログラムの高性能化を実現することを示す。まず2章でアーキテクチャの特徴について述べ、3章でアーキテクチャの特徴をコンパイラがいかに利用するかについて述べ、最後に4章でコンパイラの評価を行なう。

2 アーキテクチャの特徴

MN10200 のアーキテクチャを決定するにあたり、機器組み込み用途であるためのハードウエアを満たすとともに、コンパイラによる生成コードの分析結果を考慮し、コンパイラの最適化によって頻出する命令の実行速度を高速化し、命令語長を小さくしている。

ここでは、コンパイラにとって特に重要なレジスタセット、命令セット、アドレシングモードに関して MN10200 アーキテクチャの特徴を述べる。

2.1 レジスタセット

MN10200 はデータ用アドレス用各4本、計8本のレジスタを持つ。

このように機能を分離したことにより、命令語中のレジスタ指定フィールドを2ビットにすることができるので、最適化コンパイラで使用頻度の高いレジスタ間演算命令や基本ロードストア命令の命令語長を1バイトに収められる。このため生成コードのサイズ縮小を図ることができる。

また、機能的には分離しているが、アドレスレジスタに対する加減比較算命令を備えるため、データ値をアドレスレジスタに保持して演算することが可能となり、レジスタの有効利用が図れる。

2.2 アドレッシングモード

MN10200 は基本的にはロードストアアーキテクチャである。そのためメモリとレジスタ間の転送が頻繁に起きるので、コンパイラが有効に利用できるアドレッシングモードを備え、生成コードの圧縮、高速化を図っている。特徴としては次の三つが挙げられる。

- オフセットのないレジスタ間接のアドレッシングモードを使用するロードストア命令は語長が1バイトで、1クロックで実行する。
- 即値及びレジスタ相対間接のオフセットは8ビットと16ビットの2つのモードを持つ。このうち使用頻度の高い8ビットオフセットを用いるロード、ストア命令は1クロックで実行でき、スタックに割り当てられた変数に高速化にアクセスできる。
- データレジスタをインデックスとするアドレッシングモードを持つ。配列の添字をデータレジスタに保持させて、配列の要素を効率よくアクセスすることができる。

2.3 命令セット

Cコンパイラが使用する命令に厳選した結果、36命令で構成される。
命令セットは次のような特徴を持つ。

- ロードストアアーキテクチャであるにもかかわらず、即値の加減比較算命令を備え、1クロックで実行する。これにより組み込み用途で頻出する即値演算を高速に実行できる。
- アドレスレジスタの2インクリメント、2デクリメント命令を備える。この語長は1バイトであり、1クロックで実行する。このためループ処理等でアドレスが順に変化する配列アクセスを効率よく実行できる。

3 コンパイラの特徴

上記のように、MN10200 アーキテクチャは、コンパイラの最適化した生成コードがコンパクトで高速に実行できるように設計されている。

このことを、スタンフォードベンチマークテスト [2] のコンパイル結果を例に用いて、共通部分式最適化、不変式最適化、誘導変数最適化という最適化手法 [3] を用いた際どのようにアーキテクチャを活用しているかを考察する。

3.1 共通部分式最適化

コードサイズ圧縮のために、アドレッシングモードの活用は有効である。しかし、アドレッシングモードを豊富に備えていても、コンパイラがそれを全て活用することは困難である。またロードストアを高速化しようとするとアドレッシングモードの種類は少ないほうがよい。

そこで MN10200 は 8ビットディスプレースメントのレジスタ間接や、データレジスタをインデックスとするレジスタ間接のアドレッシングモードを備え、それらを本コンパイラの共通部分式の最適化によって有効に活用する。

スタンフォードベンチマークテスト中の Towers 関数を実行する際に必要な Push 関数からその効果が表れる部分について考察する。C ソースからの抜粋を以下に示す。

```
struct    element {
int discsize;
     int next;
};

int    stack[4];
struct element    cellspace[19];

Push(i,s)
int i, s;
{
        int localel;

        localel=Getelement();
        cellspace[localel].next=stack[s];
        stack[s]=localel;
        cellspace[localel].discsize=i;
}
```

このソースに対する生成コードは大域的最適化を行なわない場合、以下のようになる。コードサイズは34バイト、実行に必要な総クロック数は51クロックである。

```
jsr    _Getelement
mov    D2,D1
add    D1,D1
mov    D1,A0
mov    @(_stack,A0),D1
mov    #_cellspace,A0
inc2   A0
mov    #4,D3
mul    D0,D3
mov    D1,@(D3,A0)
add    D2,D2
mov    D2,A0
mov    D0,@(_stack,A0)
```

```
mov   #4,D1
mul   D1,D0
mov   D0,A0
mov   A1,@(_cellspace,A0)
```

これに対して最適化を行なうが8ビットディスプレースメントのレジスタ間接のアドレッシングモードを適用しない場合、生成コードは以下のようになる。cellspace[localel] のアドレスと stack[s] のアドレスを共通部分式として括り出した結果、コードサイズは２２バイト、実行に必要な総クロック数は３０クロックとなる。

```
mov   #_stack,A1
jsr   _Getelement
mov   #4,D1
mul   D0,D1
mov   #_cellspace,A0
add   D1,A0
mov   A0,A2
inc2  A2
mov   @A1,D1
mov   D1,@A2
mov   D0,@A1
mov   D3,@A0
```

さらに上述のアドレッシングモードを適用して最適化した場合、生成コードは以下のようになる。括りだした cellspace[localel] のアドレスに上記アドレッシングモードを適用して cellspace[localel] .next と cellspace[localel] .discsize に値を書き込んだ結果、コードサイズは２０バイト、実行に必要な総クロック数は２７クロックとなる。

```
mov   #_stack,A1
jsr   _Getelement
mov   #4,D1
mul   D0,D1
mov   #_cellspace,A0
add   D1,A0
mov   @A1,D1
mov   D1,@(2,A0)
mov   D0,@A1
mov   D3,@A0
```

アドレッシングモードを適用するとサイズで２バイト、クロック数で３クロック減少する。さらに、アドレス計算のためのアドレスレジスタが１本不要になる。８ビットディスプレースメントのレジスタ間接のアドレッシングモードの有効性がわかる。

3.2 不変式最適化

従来不変式最適化はループ処理の高速化が主な目的であった。しかしその他にレジスタを有効利用できる利点もある。スタンフォードベンチマークテスト中の Perm 関数を実行する際に必要な Permute 関数から効果の現れる部分について考察する。C ソースからの抜粋を以下に示す。

```
int     permarray[11];

Permute()
{
    for( k = n-1; k >= 1; k-- ){
        Swap(&permarray[n],‥);
        /* n はループ内で不変 */
              :
    }
}
```

このソースに対する生成コードは、大域的最適化を行なわない場合、以下のようになる。ループ処理で使用するレジスタは４本である。

```
L1
    mov   #_permarray,A0
    mov   D3,D0
    add   D0,D0
    add   D0,A0
    mov   A0,@(2,A3)
    :     :
    jsr   _Swap
    :     :
    bge   L1
```

これに対して、最適化した場合では以下のような生成コードになる。&permarray[n] の計算をループの前で行なう。ループ処理で使用するレジスタは2本である。

```
    mov   #_permarray,A0
    mov   D3,D0
    add   D0,D0
    add   D0,A0
    :     :
L1
    mov   A0,@(2,A3)
    :     :
    jsr   _Swap
    :     :
    bge   L1
```

このように、不変式最適化を行なうとループ処理の中で式の値を計算する必要がなくなり、使用するレジスタ数が少なくなる。そのため、他の変数へのレジスタの割り付けの効率がよくなり、コードサイズ圧縮を図れる。不変式最適化を行なうことで、レジスタセットを十分に活用し、効率のよいコード生成を行なえることがわかる。

3.3 誘導変数最適化

誘導変数最適化を行なうと、誘導変数を増加、減少させるコードがループ内に新たに生成される。MN10200 では2インクリメント、2デクリメント命令を用意してこのような命令の高速化を図っている。スタンフォードベンチマークテスト中の perm 関数を実行する際に必要な Initialize 関数について考察する。C ソースからの抜粋を以下に示す。

```
int     permarray[11];

Initialize ()
{
        int i;

        for ( i = 1; i <= 7; i++ ) {
                permarray[i]=i-1;
        };
};
```

このソースに対する生成コードは、大域的最適化を行なわない場合、以下のようになる。コードサイズは17バイト、ループ処理1回の実行に必要なクロック数は10クロックである。

```
    mov   #1,D1
L1
    mov   #_permarray,A0
    mov   D1,D0
    add   D0,D0
    add   D0,A0
    mov   D1,D0
    dec   D0
    mov   D0,@A0
    inc   D1
    cmp   #7,D1
    ble   L1
```

これに対して、最適化を行なうが2インクリメント命令がない場合は以下のような生成コードになる。permarray[i] のアドレスと i-1 を i の誘導変数としてコードの変換を行なう。コードサイズは15バイト、ループ処理1回の実行に必要なクロック数は5クロックである。

```
        mov   #0,D0
        mov   #_permarray+2,A0
L1
        mov   D0,@A0
        mov   #2,D1
        add   D1,A0
        inc   D0
        cmp   #6,D0
        ble   L1
```

2インクリメント命令を使用して最適化した場合は以下のような生成コードになる。コードサイズは12バイト、ループ処理1回の実行に必要なクロック数は3クロックである。

```
        mov   #0,D0
        mov   #_permarray+2,A0
L1
        mov   D0,@A0
        inc2  A0
        inc   D0
        cmp   #6,D0
        ble   L1
```

最適化した場合、2インクリメント命令がなければ、即値2をレジスタに転送して、このレジスタを使用して誘導変数の増加を行なう。そのためコードサイズで3バイト、ループ処理1回の実行クロック数で2クロック差があり、この命令の有効性がわかる。

4 システム評価

前節で述べたスタンフォードベンチマークの perm, bubble, quick, towers 関数とそれらを実行する際に必要な関数をコンパイルした生成コードに対して以下のような評価を行なった。

4.1 動的命令解析

生成コードから実行形式オブジェクトを作成し、これを実行したさいに処理した命令の数をクロック数別に集計した。表1はその結果である。

表1 命令実行頻度

	命令クロック	出現回数（単位：個）	比率
perm	1 clock	1029381	54.81%
	2 clock	640476	34.10%
	3 clock	20473	1.09%
	4 clock	93833	5.00%
	5 clock	93813	5.00%
	13 clock	10	0.00%
	合計	1877986	100.00%
bubble	1 clock	967888	73.66%
	2 clock	334713	25.47%
	3 clock	7440	0.57%
	4 clock	1200	0.09%
	5 clock	1185	0.09%
	12 clock	1000	0.08%
	13 clock	516	0.04%
	合計	1313942	100.00%
quick	1 clock	761086	57.88%
	2 clock	448804	34.13%
	3 clock	47640	3.62%
	4 clock	18999	1.44%
	5 clock	18984	1.44%
	12 clock	10000	0.76%
	13 clock	9416	0.72%
	合計	1314929	100.00%
towers	1 clock	1212061	60.41%
	2 clock	548841	27.36%
	3 clock	16152	0.81%
	4 clock	81956	4.08%
	5 clock	81955	4.08%
	12 clock	65319	3.26%
	合計	2006284	100.00%

全ての場合について、1クロック命令が50%以上を占める。1、2クロック命令合わせれば、約90%を占める。本コンパイラは実行速度向上を図るコード生成をしていることがわかる。

4.2 静的命令解析

生成コードに出現する命令の数を基本命令語長別に集計した。その表2はその結果である。

表2 命令出現頻度

	基本命令語長	出現回数（単位：個）	比率
perm	1 byte	70	82.35%
	2 byte	15	17.65%
	合計	85	100.00%
bubble	1 byte	101	80.80%
	2 byte	24	19.20%
	合計	125	100.00%
quick	1 byte	146	78.49%
	2 byte	40	21.51%
	合計	186	100.00%
towers	1 byte	176	83.41%
	2 byte	35	16.59%
	合計	211	100.00%

基本命令語長1バイトのものが８０%程度を占める。本コンパイラが生成するコードにおいて頻出する命令の基本命令語長を1バイトにするように、MN10200のアーキテクチャが設計されていることがわかる。

5 おわりに

以上、機器組み込み用マイコン MN10200のアーキテクチャの特徴と、その特徴を生かすように設計されたコンパイラの評価について述べた。今後はさらにコンパイラの最適化機能を向上させるとともに、より高性能なプロセッサアーキテクチャを設計していく予定である。

参考文献

[1] 檜垣、松崎、出口:制御用マイクロプロセッサのアーキテクチャの一考察、情報処理学会第４３回全国大会

[2] M.A.Linton :Benchmarking Engineering Workstations , IEEE Design & Test June 1986 pp.25-30

[3] A.V.Aho, R.Sethi, and J.D.Ullman: Compilers Principles, Techniques, and Tools (1988)